The Efficiency of Network Coding in P2P Content Distribution Systems

J.L.W. Brants
(1041541)

Committee members:
Supervisors: dr.ir. F.A. Kuipers, dr.ir. J.H. Weber
Members: prof.dr.ir. P. Van Mieghem, ir. J.R. Taal

September 14, 2007
Abstract

Inspired by contradictory results from the research community, we have investigated the bandwidth efficiency of network coding in a content distribution system. We have developed a simulation model for a P2P network consisting of (many) nodes and one registrar. Each network starts with at least one node that has all distributable content – a seed. Nodes can exchange information with a small neighbor subset and report their statistics and requests for new neighbors to the registrar. From our simulations we can conclude that the bandwidth efficiency of network coding in a P2P content distribution system varies between 98 and 99%. Small fluctuations are caused by (1) the network size, (2) the number of links to neighbors per node (it must have at least 3, preferably 4), (3) the seed’s bandwidth (as long as it is at least equal to the nodes’ bandwidth), and (4) dropped links (e.g., caused by nodes joining and leaving the network).

We have proposed two new coding algorithms: delta network coding and smart network coding. Delta network coding performs better in terms of bandwidth efficiency and complexity to the existing algorithms complete network coding and sparse network coding. It does come with a small delay between the reception and decoding of vectors, which could make it less interesting to time-critical applications, such as streaming video. Smart network coding offers a tunable trade-off between complexity and efficiency.

When we compare our results with recent studies that carry out content distribution system simulations of Bittorrent and naive broadcast, we can conclude that network coding outperforms both other protocols.
# Contents

1 Introduction .................................................. 5  
1.1 Peer-2-Peer Content Distribution ......................... 6  
1.2 Network Coding ............................................. 8  
1.3 Problem Statement ......................................... 11  

2 Model Description ............................................. 12  
2.1 Network Layer ............................................... 12  
2.2 Application Layer .......................................... 14  
2.3 Performance Indicator ....................................... 15  

3 Random Coding Algorithms .................................... 17  
3.1 Complete Network Coding ................................... 17  
3.2 Sparse Network Coding ...................................... 19  
3.3 Delta Network Coding ....................................... 22  
3.4 Smart Network Coding ...................................... 23  

4 Performance Analysis ......................................... 25  
4.1 Basic Properties ............................................. 25  
4.2 Reliability ................................................... 28  
4.3 Optimal Sparsity $\sigma$ ...................................... 31  
4.4 Optimal Randomness $\rho$ .................................... 32  
4.5 Coding Algorithms Comparison ............................. 33  
4.6 Comparison with BitTorrent and Naive Broadcast ........ 36  

5 Implementation Issues ......................................... 38  
5.1 Complexity .................................................. 38  
5.2 Spread Delay ................................................ 39  
5.3 Incentive Mechanisms ....................................... 40  
5.4 Security ...................................................... 40  

6 Conclusions .................................................... 42  
6.1 Conclusions ................................................ 42  
6.2 Directions for Further Research ............................ 43  

A The Hopcount in Random Regular Graphs ...................... 44  
Nomenclature ................................................... 44  
Bibliography .................................................... 47
Chapter 1

Introduction

In recent years peer-2-peer (P2P) systems have become very popular for content distribution. A study from 2004 [5] estimates that 80% of all Internet traffic can be credited to P2P technology based programs. First mainly used to exchange illegal content (e.g., KaZaA, eDonkey, Bittorrent) P2P technology now attracts more and more attention from legal content distributors (such as Skype\textsuperscript{1}, Joost\textsuperscript{2}, BBC iMP\textsuperscript{3}, and Tribler\textsuperscript{4}). Its power lies in the fact that, contrary to traditional client-server content distribution systems, the distributor only needs limited upload bandwidth, because downloading clients act at the same time as uploading servers to other users. Depending on the number of simultaneous users, P2P technology can reduce the required upload bandwidth to only a fraction of that with the traditional distribution method.

C. Gkantsidis et al. [13] were the first to use network coding in P2P systems. They claimed that their system outperforms traditional P2P systems by at least a factor two. Later work [11, 12] subscribed to these conclusions. However, M. Wang and B. Li [21] compared the network coding method with naive broadcast (which they consider the worst possible protocol) and concluded that the latter performs better. This discrepancy between research results led to the motivation for our study.

We shall first explain the principles of P2P content distribution and network coding, before we give the problem statement and an outline of our report.

\textsuperscript{1}http://www.skype.com/
\textsuperscript{2}http://www.joost.com/
\textsuperscript{3}http://www.bbc.co.uk/imp/
\textsuperscript{4}http://www.tribler.org/
1.1 Peer-2-Peer Content Distribution

**Overlay network**  P2P technology makes use of an overlay network. Nodes in the P2P content distribution environment create this overlay network by connecting to several other nodes in the environment using an existing network (e.g., the Internet). These connections together form a new network, called an overlay network, shown schematically in Figure 1.1.

**Fragment scheduling**  The most important property of a P2P protocol is fragment scheduling. The data that is to be distributed is split into (usually several hundred) fragments. Each node tries to collect these fragments from its neighbors to recreate the original file (see Figure 1.2a). Downloading stalls when a node already possesses the fragments of all its neighbors (node C in Figure 1.2b).

Existing P2P protocols try to prevent this from happening by using a fragment scheduling method called local-rarest-first. Nodes ask their neighbors about the fragments they have and will first download the fragments that are locally most rare. However, because nodes do not have a total overview of the shortage of fragments in the whole network, this does not guarantee perfect fragment scheduling. A fragment could be locally rare while most other nodes already possess it, or a fragment could be locally available while it is rare for most other nodes.

Fragment scheduling is not only important to guarantee fast downloads, it is also needed to prevent incentive mechanisms from stalling the distribution of the fragments. Incentive mechanisms are developed to prevent free-riders from only downloading fragments while not uploading any. They all use a form of the tit-for-tat principle: I give you a fragment you do not have, when you give me a fragment I do not have. For this mechanism it is not only important that neighbors have fragments a node does not have, but also that the node has fragments its neighbors do not have. In Figure 1.2c we see that node A could download a fragment from node B, but since it does not have anything interesting in return, no exchange takes place.
**Bittorrent**  The most used P2P file exchange protocol today is Bittorrent, developed by B. Cohen. It makes use of an overlay network where each node has 8-16 neighbors. Local-rarest-first is used for fragment scheduling, except at the start and end of a download. At the start a node downloads random fragments, until it completes a fragment. At the end of the download an endgame mode is used: when all missing fragments are being downloaded, it still sends out requests for those fragments.

Bittorrent uses “choking” as an incentive mechanism. A node only sends fragments to a fixed number of neighbors (default is 4) that provide the best download rates. To be able to discover neighbors that can offer higher download rates, one random node is “optimistically unchoked” every 30 seconds. For more details see [7].

**Performance indicator**  Because we want to be able to say something about the efficiency of a content distribution system, we wish to make a comparison of the performance with the best possible performance.

One of the most important performance indicators for the end-user is: “how long does it take to finish my download?” To answer this question we could obviously measure the time between the start and finish of a download. Unfortunately, a problem arises when we want to compare this with the lowest possible download time. With a client-server system the lowest possible download time can be calculated straightforward from the file size and the available bandwidth. However, in a P2P network, fragments must spread through the network, which means that there is a delay between the broadcast of a fragment by the source, and the reception of this fragment by all other nodes. The calculation of this spread delay is complex, and becomes even more complex when we consider a dynamic network. This complexity makes the download time less appropriate as a performance indicator.

The used bandwidth $B_{\text{used}}$ is very closely related to the download time $T$ as

$$S = \int_0^T B_{\text{used}} \, dt$$

---

---

Figure 1.2: Fragment exchange
where $S$ is the total size of the data. We can approach the highest possible available bandwidth with $B_{W_{eff}}$, the bandwidth that is effectively available to a node. This bandwidth considers only the bandwidth that is offered by the neighbors of the node that possess at least one fragment. The bandwidth efficiency can then be defined as

$$BE = \frac{B_{W_{used}}}{B_{W_{eff}}} \cdot 100\%$$

Wrong fragment scheduling would in this case lead to a lower efficiency, but an increase in the spread delay will not affect the efficiency.

This approach does still introduce some errors when there exist cycles in the P2P network. Consider the case where node $i$ is part of a cycle and receives and forwards its first fragment. When the other nodes on the cycle do not receive any other fragments, except the one from $i$, they have no other choice than to forward this fragment. As the cycle has a limited length, eventually the fragment comes back to $i$. However, it can never be of interest to node $i$ and an inefficiency is introduced. Obviously, there does not exist a way in which the other nodes on the cycle could have sent something useful to node $i$ as they did not receive any other fragments yet, and therefore our bandwidth efficiency approach can introduce false inefficiencies. Unfortunately we cannot distinguish this false inefficiency from the case where a node on the cycle did receive another fragment and it just forwarded the “wrong” fragment.

### 1.2 Network Coding

**General principle** Traditionally, to perform multicast in a network (consider Figure 1.3 where node A tries to send messages $M_1$ and $M_2$ to nodes F and G), nodes make use of the store-and-forward method: an intermediate node stores all received messages and forwards those to its neighbors. Which of the stored messages is replicated depends on the awareness of the node: it may simply pick a message randomly or ask its neighbors which message they do not have yet. However, even with a perfect awareness of which node possesses which messages, this method may not reach optimal performance. In Figure 1.3a, the problem lies between node D and E: which message should be sent to node E? Sending $M_1$ would give node G both messages but would leave node F with only one message, sending $M_2$ leaves node G with only one message. Clearly this problem cannot be solved with traditional forwarding methods.

R. Ahlswede et al. [1] proposed to allow nodes to combine their received messages into one or more outgoing messages. We see in Figure 1.3b that, when we binary add $M_1$ and $M_2$, this does indeed solve our problem: node F just needs to binary subtract $M_1$ from the message received from E to
find both messages back, and similarly for node G. This simple but effective technique is called “Network Coding” and an overview of its applications is given in [10].

In the previous network coding example, the simplest form of linear combining is used: binary addition and subtraction, also known as the XOR operation. However, combining messages is possible over any finite field $\mathcal{F}$, also known as Galois fields, which provides us with specific mathematical operations for addition and multiplication.

**Encoding** To create encoded messages we first split our original data $\mathbf{M}$ into $V$ vectors $\mathbf{M}_1, \ldots, \mathbf{M}_V$. We obtain an encoded message by multiplying each original vector with a coefficient $r_i$: $\mathbf{I} = \sum_{i=1}^{V} r_i \mathbf{M}_i$. From now on we shall call the encoded message $\mathbf{I}$ the information vector and the coefficients $\mathbf{E} = [r_1 \ldots r_V]$ the encoding vector.

Because of the linear property, encoding can be performed recursively. When a node has received $b$ information vectors $\mathbf{I}_i$ with corresponding encoding vectors $\mathbf{E}_i$, it can create a new information vector $\mathbf{I}' = \sum_{i=1}^{b} r_i \mathbf{I}_i$ with a corresponding encoding vector $\mathbf{E}' = \sum_{i=1}^{b} r_i \mathbf{E}_i$. The coefficients $\mathbf{R} = [r_1 \ldots r_b]$ that the node uses are often called the local encoding vector and the resulting coefficients $\mathbf{E}'$ are called the global encoding vector. The calculated $\mathbf{I}'$ and $\mathbf{E}'$ are new $\mathbf{I}_i$ and $\mathbf{E}_i$ for neighboring nodes, who can again perform recursive encoding.

**Decoding** For any node to retrieve the original data from the received information and the global encoding vectors, it needs to solve the system.
\[
\begin{bmatrix}
I_1 \\
\vdots \\
I_V
\end{bmatrix}
= 
\begin{bmatrix}
E_1 \\
\vdots \\
E_V
\end{bmatrix}
\begin{bmatrix}
M_1 \\
\vdots \\
M_V
\end{bmatrix}
\]

which is only possible when
\[
\begin{bmatrix}
E_1 \\
\vdots \\
E_V
\end{bmatrix}
\]
is a full rank matrix. This is only the case when we have received \( V \) independent encoding vectors \( E_i \): any received dependent encoding vector is useless and can be dismissed. Newly received, independent vectors are called innovative.

**Coefficients** When the network topology is known, the local coefficients leading to optimal performance can be found in polynomial time [15]. However, in a P2P environment it is not likely that the topology is known at all times, and recalculating them each time the topology changes would also cost a lot of time. P. Chou et al. [6] proposed to select the coefficients of the local encoding vectors randomly, calculate the corresponding global coefficients and send those together with the information vector. They call this random network coding. The probability that a non-innovative vector is created then depends on the field size \( |F| \) as [9]

\[
Pr(\text{non-innovative vector}) \leq \frac{1}{|F|} \tag{1.1}
\]

Thus, when we select an appropriate field size, this probability becomes negligible. Simulation results [24] show that even a small field size such as \( 2^8 \) already suffices. As random network coding offers good flexibility with respect to topology changes at the (small) cost of some overhead (the encoding vector needs to be sent together with the information vector) this is the coding type of choice in all network coding content distribution systems so far. We shall use it as well.

**Theoretical benefits and disadvantages** In a network coding environment each vector carries equal importance. When we choose our coefficients such that each vector is independent, every node just needs any \( V \) vectors to be able to decode. Any offered vector is of interest to all nodes which means that we no longer have to schedule fragments using e.g. local-rarest-first. Moreover, as shown in Figure 1.3, under certain circumstances network coding can outperform any choice a store-and-forward system can make.

Not only does this advantage come at the cost of some overhead, network coding also requires a lot more computing power than store-and-forward, as
new information vectors have to be encoded and received information vectors have to be decoded.

1.3 Problem Statement

C. Gkantsidis et al. [13, 11, 12] proposed a content distribution system based on random network coding. From simulations [13] they found that “The performance benefits provided by network coding in terms of throughput are more than 20-30% compared to coding at the server, and can be more than 2-3 times better compared to transmitting unencoded blocks.” B. Cohen responds to the claim that the outperformed uncoded blocks system is very similar to Bittorrent, stating that “intentionally or not, the simulation is completely rigged against Bittorrent.” In later work Gkantsidis et al. [11, 12] did not make any comparisons, but found that the implementation they had made highly utilized the system resources and provided the user with a smooth download progress.

M. Wang et al. [21] were “more cautious and less optimistic than previous studies” as they found that “a coded peer-to-peer session offers poorer performance in terms of downloading times, as compared to the worst possible non-coding protocol, naive broadcast.” They believed that this was due to the fact that nodes had to collect several vectors, before they could start serving their neighbors, in order to prevent that they would send non-innovative vectors.

In this report we aim to give more insight in the performance of network coding used in a P2P content distribution system under several circumstances, in such a way that it can be compared with the performance of other methods of P2P content distribution. A comparison with the performance of Bittorrent and naive broadcast hopefully provides us with more insights whether network coding truly can outperform Bittorrent, or indeed has a very poor performance.

A description of the model for a P2P system in which our research takes place is given in Chapter 2. In Chapter 3 we discuss existing random coding algorithms, and propose two new algorithms that could reduce the complexity of network coding. Chapter 4 shows and discusses the performance of network coding under several circumstances. Chapter 5 discusses some issues that are not covered in our model and finally we conclude our work and give some directions for future work in Chapter 6.
Chapter 2

Model Description

We have chosen to create a model for a P2P system in C++, as this enables us to perform simulations with large numbers of nodes within reasonable times. The model, schematically depicted in Figure 2.1, consists of two entities: one or more nodes and a registrar. The nodes buffer received vectors and forward new vectors to other nodes. Nodes can only receive from and forward to nodes who are their neighbors, a limited-size subset randomly chosen from the total set of nodes. In Figure 2.1 neighboring nodes are indicated with solid lines. The seed is the source of all vectors – it is a node that already possesses all data. Nodes periodically report their statistics to the registrar. The registrar facilitates node-linking and collects the nodes’ statistics (Figure 2.1: the dashed lines).

2.1 Network Layer

Our model only considers the P2P overlay network: it assumes that the upload bandwidth of the Internet connection of each node is the limiting

Figure 2.1: Schematic view of our model for a P2P system.
factor. Delays between connected nodes are not considered, and because all links are reciprocal and nodes usually have an asymmetric bandwidth distribution in favor of the download bandwidth, the download bandwidth of each node is limited by its upload bandwidth. Each link has unit bandwidth and therefore a node’s upload/download bandwidth is determined by its number of links $L$.

At the start of each simulation the number of nodes $N$ that will be simulated are initialized. In a random order nodes that still demand more neighbors (meaning that their maximum number of links $L$ has not yet been reached) are interconnected by the registrar. One node is made seed. The obtained connectivity graph $G(N, L)$ is of the type random regular graph, which has the following property [23]

**Theorem 2.1.1** If $L \geq 3$ and fixed, then $G(N, L)$ is asymptotically almost sure (a.a.s.) $L$-connected.

We call a graph $L$-connected when between any pair of nodes $i$ and $j$ there exists a path connecting them, in any subgraph obtained by deleting maximally $L - 1$ links. Because all links have unit bandwidth, the minimum cut of our network is then equal to $L$. Combining this result with the max-flow min-cut theorem [3], which states that the maximum flow equals the minimum cut between any two nodes, we find an important corollary for our overlay network

**Corollary 2.1.2** If $L \geq 3$ and fixed and all links have unit bandwidth, the maximum flow between any two nodes in $G(N, L)$ is a.a.s. equal to $L$.

To facilitate the joining of new nodes during a simulation, nodes have the possibility to randomly drop a link. A node that drops a link and informs the other end of the link (its former neighbor). Both will then report to the registrar that they are in demand of a new neighbor. As a new node tries to enter the network, the registrar will connect it with nodes that are still looking for a neighbor. When this is not sufficient to connect all links of the new node, it will tell a random node to drop a link, and connect it to the new node. This process is repeated until the new node has connected all of its links with the network.

Nodes also have the possibility to leave the network. When they do this, they inform all neighbors that they will drop their links. When a node has left the network, it will no longer inform the registrar that it is in demand of new neighbors.

---

1 A property $E_N$ is considered to be a.a.s. when $\Pr(E_N) \to 1$ as $N \to \infty$
2.2 Application Layer

A node accepts all incoming vectors – even when they are non-innovative. The following phases are iterated until all nodes have finished downloading.

1. All nodes send newly created vectors to all of their neighbors. Figure 2.2: e.g., node 1 sends its vectors [4 1 2] and [1 3 1] combined into a new vector [5 4 3] to node 3. The receiving node places these vectors in its process buffer. E.g., node 3 places received vectors [5 4 3] and [7 1 5] in its process buffer.

2. After all nodes have finished sending, they process the received vectors from the process buffer, to see if they are innovative. If they are, they are saved in the received vectors buffer, otherwise they are discarded. E.g., node 3 finds that it already has vector [5 4 3] and discards it. It did not have vector [7 1 5] yet, so this vector is placed in the received vectors buffer.

3. The registrar is updated with the nodes’ performance indicator (explained in Section 2.3) and takes care of the possible node-linking that would be required in the case of nodes leaving or joining the network.

The three phases together are called a timestep.

To maintain a high simulation speed, the most important property of the model is that only the encoding vectors are actually sent and buffered. The information vectors that would normally be transferred with them, give
no extra information about whether a node is able to decode the original information or not, and is therefore omitted.

The computational gain we achieve with this choice, gives us the opportunity to investigate much bigger P2P networks than previously has been done. Where [12, 21] simulate up to 100 nodes, we have managed to simulate up to 3000 nodes. Even higher numbers can be achieved, but our results indicated that such simulations would not lead to more insights.

The disadvantage of this choice is that no information is gained about the influence of the size of the information vectors. It makes it also impossible to say anything about the computing power that would be required to perform the coding operations in an implementation (where one would of course require the actual data to be sent as well). However, we believe that these influences have been researched thoroughly in several other papers, such as [21, 22].

Both the algorithm used for the choice of the random coefficients $r_i$, and the algorithm used to decode the received vectors $E_i$ depend on the coding algorithm, and are explained in detail in Chapter 3. Computations are done in a $GF(2^{16})$ module.

### 2.3 Performance Indicator

To measure the performance of the nodes, and with that of the P2P system, we shall use the efficiency indicator as mentioned in Section 1.1: a comparison of the used bandwidth with the effectively available bandwidth. When $V_{inn}$ are the number of innovative vectors and $V_{tot}$ are the total number of vectors, received in the download time $t$, the used bandwidth $B_{W_{used}}$ is

$$B_{W_{used}} = \frac{V_{inn}}{t}$$

and the effectively available bandwidth $B_{W_{eff}}$ is

$$B_{W_{eff}} = \frac{V_{tot}}{t}$$

giving us a formula for the bandwidth efficiency

$$BE = \frac{V_{inn}}{V_{tot}}$$

provided that each node sends a vector to its neighbors as soon as it possesses at least one vector itself, whether this vector is innovative for the receiving node or not. As the last requirement is the case for the nodes in our model, we can use this performance indicator.
Error estimation  As we have seen in Section 1.1, the use of the bandwidth efficiency as a performance indicator introduces false inefficiencies. We try to give an idea of the dependence of the error on the parameters of the model. We use [4]

**Theorem 2.3.1** For $L$ fixed, let $X_i = X_{i,N}$ ($i \geq 3$) be the number of cycles of length $i$ in a graph $\mathcal{G}(N,L)$. For fixed $k \geq 3$, $X_3, \ldots, X_k$ are a.a.s. independent Poisson random variables with means $\lambda_i = \frac{(L-1)^i}{2i}$.

This Theorem shows that the number of cycles increases with $L$. As the number of cycles is asymptotically stable, the number of cycles per node will decrease as $N$ increases. Finally the error a cycle causes depends on the hopcount of the network. As we have seen in Section 1.1, for the inefficiency to be false, none of the nodes on the cycle have to receive another vector before forwarding the vector that will cause the inefficiency. When the length of a cycle is smaller than the hopcount, it is likely that this can happen. As the hopcount depends negatively on $L$ and positively on $N$, the error caused by a cycle with length $i$, will have a peak at some $L$ and some $N$. As the location of the peak depends on the cycle length $i$, we can expect to see several peaks in the number of false inefficiencies as we vary $N$ or $L$. We have visualized this in Figure 2.3.
Chapter 3

Random Coding Algorithms

Because in a network coding environment every node has to encode and decode vectors, the required processing power is higher than in traditional store-and-forward P2P systems. We explain the original random coding algorithm, which we shall henceforth refer to as complete network coding. We seek for random coding algorithms achieving similar efficiencies as complete network coding, but at lower coding complexities. We discuss both an existing algorithm: sparse network coding, and two new algorithms: delta network coding and smart network coding. For all algorithms we give both the basic principles and a formula for (an upper bound of) the coding complexity.

3.1 Complete Network Coding

**Principle**  The original random coding principle states that we should encode new vectors with a local encoding vector \( \mathbf{R} \) where its coefficients \( r_i \) are randomly picked from \( \mathcal{F}\{0\} \). According to Equation 1.1, when we choose the field size large enough, the probability for non-innovative vectors becomes negligible.

To retrieve our original data \( \mathbf{M}_1 \ldots \mathbf{M}_V \) from the received global encoding vectors \( \mathbf{E}_i \) and information vectors \( \mathbf{I}_i \), we have to bring the left part of the matrix

\[
\begin{bmatrix}
\mathbf{E}_1 & \mathbf{I}_1 \\
\vdots & \vdots \\
\mathbf{E}_V & \mathbf{I}_V
\end{bmatrix}
\]

into the reduced echelon form using row operations, so that \( \mathbf{M}_i = \mathbf{I}_i \). We use the Gaussian elimination algorithm for this process, which considers the rows \( \mathbf{E}_i \) in order of arrival. The algorithm reduces a non-innovative row to zero, so this way we can see immediately after the arrival of a vector, whether it is innovative or not.
The fact that the information vectors $I_i$ are not sent in our simulation, does not influence the required row operations, and thus it does not affect the outcome of the Gaussian elimination. It does however reduce the required number of operations within a row operation, which reduces the computing complexity of the model. Because we want to compare the complexities of the different algorithms, would they be implemented, we do consider the information vectors in our complexity calculations.

**Complexity** When we define our standard operation as $a + (b \cdot c)$ or $a - (b \cdot c)$, where $a$, $b$, and $c$ are elements of $F$, we can give the complexity of a coding algorithm as the number of required standard operations. The first received row is already in echelon form and thus needs no operations. To bring the second row into echelon form we need to subtract a multiplication of the first row from it, which requires $(V + l)$ standard operations, where $l$ is the length of the information vectors. Under the assumption that $V \ll l$, we need approximately $l$ operations. The third column now requires $2l$ operations, the fourth $3l$, etc. The length $l$ of each information vector is simply the total length of the original data divided by the number of vectors it is split into, so

$$l = \frac{S}{V}$$

with $S$ the total size of our data $M$. The number of operations needed to bring all vectors into echelon form is then

$$\sum_{i=1}^{V} (i - 1) \cdot l = \frac{(V - 1)V}{2} \cdot l = \frac{S(V - 1)}{2}$$

To bring the matrix from the now obtained echelon form into the required reduced echelon form, we need to repeat the exact same steps in reversed order. The total decoding complexity is then given as

$$C_d = S(V - 1)$$

When we assume that after each received vector a new vector is transmitted using all available received vectors, encoding the first vector requires $l$ computations, the second $2l$, the third $3l$, etc. The encoding complexity is then given as

$$C_e = \sum_{i=1}^{V} i \cdot l = \frac{V(V + 1)}{2} \cdot l = \frac{S(V + 1)}{2}$$

Hence the complexity of both encoding and decoding is of order $V \cdot S$ and thus linear in both the number of vectors $V$ and the size of the data $S$. When we assume $V \gg 1$ we find the total complexity

$$C = C_d + C_e \approx \frac{3SV}{2} \quad (3.1)$$
3.2 Sparse Network Coding

**Principle** In [18] Guanjun Ma et al. propose to use a sparse coding strategy: only a part of all received vectors are used to create new vectors, which reduces the encoding complexity. The decoding process makes use of Gaussian elimination, just like complete network coding. They claim that this strategy should work based on a theorem, deduced from [8]. They deduced

**Theorem 3.2.1** When given a random $V \times V$ matrix $A = (a_{ij})$ where each entry over finite field $F$ is identically independently distributed with

$$\Pr(a_{ij} = r) = \begin{cases} 1 - p & r = 0 \\ p/(|F| - 1) & r \in F \setminus \{0\} \end{cases}$$

then

$$\lim_{V \to \infty} \Pr(A \text{ is non-singular}) = e^{-2e^{-d}} \prod_{j=1}^{\infty} \left(1 - |F|^{-j}\right)$$

provided that

$$p \geq \frac{\log V + d}{V}$$

Hence, when we define the global encoding matrix as

$$A = \begin{bmatrix} E_1 \\ \vdots \\ E_V \end{bmatrix}$$

where the $E_i$ are the global encoding vectors the seed sends, the probability that the seed creates non-innovative vectors is still negligible when it only encodes $M_i$ with a probability $p$, provided that we choose an appropriate $d$ and use a sufficiently large number of vectors $V$. For example, for a sufficiently large $V$ and $|F| = 2^{16}$, choosing $d = 8$ would lead to a non-singular probability 99.99%, and requires $p$ to be 0.1. As now only one out of every ten $M_i$ is used in the encoding process, this would reduce the seed’s encoding complexity by 90%.

Theorem 3.2.1 is applicable for the seed, which starts having all $V$ vectors. We must however, account for the fact that non-seed nodes receive the $V$ innovative vectors over time. They create new vectors from the received vectors $E_i$ and $I_i$ and thus we can write their global encoding matrix as the product of their local encoding vectors $R_i$ – which together form a local encoding matrix $A_L$ – with the received vectors:

$$A = \begin{bmatrix} R_1 \\ \vdots \\ R_V \end{bmatrix} \times \begin{bmatrix} E_1 \\ \vdots \\ E_V \end{bmatrix} = A_L \times \begin{bmatrix} E_1 \\ \vdots \\ E_V \end{bmatrix}$$
Only the product of two non-singular matrices is again a non-singular matrix and hence $A_L$ has to be non-singular for $A$ to be non-singular.

For this, Guanjun Ma et al. state that it should be sufficient when these nodes construct their $A_L$ similar to $A$ from Theorem 3.2.1, except that

$$p = \frac{\log V + d}{V_{inn}}$$

with $V_{inn}$ the number of received innovative vectors at the node. Thus, the more innovative vectors a node has received, the smaller the probability gets that a vector is used in the encoding process of a new vector.

We believe that this is not generally sufficient. Consider the network of Figure 3.1 where a seed $s$ is connected to a node $i$ that is connected to another node $j$. When node $i$ has only one link to the seed, as in Figure 3.1a, for the $q^{th}$ new vector it creates, it can only use the $q$ vectors it has received so far from the seed (assuming the seed sends only innovative vectors). Then $V_{inn} = q$ and the local encoding matrix $A_L$ will be triangular. A triangular matrix can only be non-singular when all entries on the diagonal are non-zero. As

$$\Pr(a_{qq} = r) = \begin{cases} 1 - p_q & r = 0 \\ p_q/(|F| - 1) & r \in F \setminus \{0\} \end{cases}$$

with

$$p_q = \frac{\log V + d}{q}$$

according to Equation 3.2, we can now write

$$\Pr(A_L \text{ is non-singular}) = 1 - \prod_{q=1}^{V} (1 - p_q) \leq \frac{\log V + d}{V}$$

for the local encoding matrix of $i$. Except when $p_q = 1 \ \forall \ q$, which is not the case for the $d$ given by Theorem 3.2.1, this probability is not negligible. Hence, using Equation 3.2 is not sufficient for the theorem to be applicable to non-seed nodes as well!

When we add an extra link between $i$ and $s$, then $V_{inn} > q$ and there can be non-zero entries to the right of the diagonal as well (Figure 3.1b). Hence, the more extra links we add to the seed, the more the local encoding matrix $A_L$ will tend towards the matrix $A$ from Theorem 3.2.1 – in which all entries $a_{ij}$ can be non-zero – and the more the probability that $A_L$ is not a non-singular matrix will become negligible.

There is also a more intuitive explanation that applies for any network where $L_{seed} = L_{node}$. When vectors travel from the seed to a node over the $L$ shortest paths to this node, there is a possibility that they get delayed, because an intermediate node does not use it in the encoding process of the next new vector (the corresponding encoding coefficient $r = 0$). As
Figure 3.1: Structure of the local encoding matrix $A_L$ of a non-seed node.

$L_{seed} = L_{node}$, all sent vectors must travel without delay over the shortest paths to a node, otherwise this node cannot receive only innovative vectors. This problem can be solved by increasing $L_{seed}$. Only $L_{node}$ of the $L_{seed}$ sent vectors have to arrive without delay at each node, to prevent non-innovative vectors. As we increase $L_{seed}$, the probability that there are at least $L_{node}$ vectors without delay, increases as well.

When nodes join at different times, oversizing $L_{seed}$ becomes less important, as a node will already have several received vectors before it has to start sending to an arrived node, and thus $V_{inn} > q$. Again this leads to more non-zero entries to the right of the diagonal, as in the situation where we have increased $L_{seed}$.

**Our implementation** The sparse coding strategy we have implemented is slightly different from the former algorithm: we deterministically use $\sigma$ vectors to encode a new vector. This sparsity $\sigma$ is strongly related to $d$ through the expected value of the number of encoded vectors, as

$$E[\text{number of encoded vectors using } d] = p \cdot V_{inn} = \log V + d \quad (3.3)$$

and

$$E[\text{number of encoded vectors using } \sigma] = \sigma \quad (3.4)$$

Hence the expected value of the number of encoded vectors is the same for both methods when $\sigma = \log V + d$. In either case we now have to find
the smallest $d$ or $\sigma$ – both leading to less encoded vectors and thus to a smaller complexity – and the smallest $L_{seed}$, that still leads to a negligible probability for a non-innovative vector. If we would use $\sigma = V$, for each new vector all received vectors are used, which is the same as complete network coding.

**Complexity** The decoding process of sparse network coding is the same as in complete network coding. Hence the complexity can be written as

$$C_d = S(V - 1)$$  \hspace{1cm} (3.5)

The number of vectors that are used to encode a vector is bounded by the sparsity $\sigma$ and the encoding complexity becomes

$$C_e = \sum_{i=1}^{V} \min[i, \sigma] \cdot l = \sum_{i=1}^{\sigma} i \cdot l + (V - \sigma)\sigma \cdot l = \sigma(2V - \sigma + 1) \cdot l = \sigma(2V - \sigma + 1)S$$ \hspace{1cm} (3.6)

### 3.3 Delta Network Coding

**Principle** As we have seen in the intuitive explanation in Section 3.2, inefficiencies arise when vectors are delayed when they travel along the $L$ shortest paths to the nodes. This would imply that when we ensure that the vectors are not delayed, we prevent inefficiencies from occuring. We could do this by using the $L$ vectors we have received at timestep $t-1$ in the encoding process of timestep $t$. This would reduce the number of vectors used in the encoding process, depending on the number of links.

However, as we wish to use the vectors as we received them in the last timestep, we cannot decode these vectors on-the-fly, with Gaussian elimination. We could still decode them after they have been used in the next timestep though.

**Complexity** The fact that we cannot use on-the-fly decoding does not influence the complexity of the decoding process, as this process is only performed one timestep later. Hence, the decoding process of delta network coding is the same as in complete network coding and can be written as

$$C_d = S(V - 1)$$ \hspace{1cm} (3.7)

The number of vectors that are used to encode a new vector is bounded by the maximum number of vectors that can be received in one timestep. Since we do not have to encode the vector we received from the node we are
going to send a vector to, the number of vectors we encode are the $L - 1$ last received vectors. The encoding complexity now becomes

$$C_e = \sum_{i=1}^{V} \min[i, L - 1] \cdot l = \frac{(L - 1)(2V - L + 2)}{2} \cdot l$$

$$= \frac{(L - 1)(2V - L + 2)S}{2V}$$

(3.8)

### 3.4 Smart Network Coding

**Principle** We can consider the Gaussian elimination algorithm that reduces received vectors to echelon form, to be a pseudo random encoding process at the same time: it adds multiples of already received vectors to the last received vector. This means that a vector reduced to echelon form is at the same time a new pseudo random vector. Combining several pseudo random vectors leads to a vector that is “random enough” and thus is as good as a new vector created from all received vectors. Sparse network coding makes use of this fact: it random-linearly combines the last $\rho$ received vectors.

Because we decode received vectors immediately to echelon form, we need to ensure that the pivot position of the sent vector is high enough for the recipient. We do this by also encoding the vector with the lowest pivot position that still is at least as high as the pivot position the recipient requires. In Figure 3.2 we see that if vectors 5 and 6 were received since the last transmission to the recipient, sending only those would give a new vector which has a too low pivot position. When we also encode vector 3 we solve this problem.

Because the pivot position of a new vector is not necessarily located at the last entry (as we have seen in Figure 3.2), the decoding process will not always use all already received vectors to bring a received vector into echelon form.
form. Therefore, we keep track of the vectors we have not used to reduce the last $\rho$ vectors to echelon form and add a random linear combination of these vectors to the new vector as well.

We now want to find the smallest possible randomness $\rho$ – leading to the smallest possible complexity – that still provides us with new vectors that are “random enough”, and thus performs as well as complete network coding. If we would use $\rho = V$, for each new vector all received vectors are used, which is the same as complete network coding.

**Complexity** When we define $N_i$ as the set of vectors that are not used to decode vector $i$, the decoding complexity becomes

$$C_d = S(V - 1) - \sum_{i=1}^{V} |N_i| \cdot l$$

Encoding a new vector requires the last $\rho$ received vectors plus one vector to ensure a high enough pivot position, and the vectors that have not been used to bring those $\rho$ vectors into echelon form. The encoding complexity is then given as

$$C_e = (\rho + 1)V \cdot l + \sum_{i=1}^{V} |N_i \cap N_{i-1} \cap \ldots \cap N_{i-\rho+1}| \cdot l$$

$$\leq (\rho + 1)S + \sum_{i=1}^{V} |N_i| \cdot l$$

We have found an upper bound for the total complexity of smart network coding with

$$C = C_d + C_e \leq S(V - 1) + (\rho + 1)S = S(V + \rho) \quad (3.9)$$
Chapter 4

Performance Analysis

We use our model from Chapter 2 to find the performance of network coding in a P2P environment under several circumstances. We first look at the performance varying some basic properties of the network, such as the number of nodes and links. After that we look at the performance when the network becomes less predictable, having properties that make it less reliable, such as leaving nodes and dynamic links. We then use our model to find the optimum sparsity $\sigma$ for sparse coding and $\rho$ for smart coding. Finally we compare all proposed coding algorithms from Chapter 3 and also make a basic comparison with Bittorrent and naive broadcast.

Our simulations have been run on a cluster consisting of 5 3Ghz P4 machines, lasting in total more than two months. All simulations have been done using 100 vectors, as this was both high enough to ensure we would not encounter big discretization problems (remember each link sends one vector every timestep), but low enough to ensure decent simulation times. We will discuss the trade-offs involved in the choice of the number of vectors in depth in Section 5.1.

The values for the mean and variance in Figures 4.1 until 4.6 were obtained using the statistics of 10,000 nodes (e.g., when a simulation consisted of 1000 nodes, the data of 10 of those simulations was used, when it consisted of 500 nodes, 20 simulations were used, etc). The mean values of the other figures were obtained averaging the statistics of 50 simulations. The variance of the obtained data is only relevant in the first two figures, and thus not depicted in the other figures.

4.1 Basic Properties

Varying the number of nodes A P2P network must preferably be very well scalable. Hence we do not want to find a negative dependence between the performance of the system and the number of nodes. As Figure 4.1 shows this dependence does indeed not exist, except for a very small number of
nodes. With less than 50 nodes in the network, a slightly higher number of non-innovative vectors is received. This can be explained by understanding that with a very small number of nodes, the requirement that $N \to \infty$ in the definition of a.a.s. is not satisfied, and Theorem 2.1.1 is no longer applicable. When the network is not $L$-connected while the seed and a receiving node have $L$ links, the minimum cut between them must be smaller than $L$. Hence, the network between the seed and node is now the limiting factor, instead of the number of links of the seed and node. As a node can now no longer make full use of its links, this leads to a decreased efficiency.

The small periodic behavior in the trend of the mean (we can see maxima at 100 and 600 nodes, and minima at 250 and 1500 nodes) is caused by errors due to cycles in the network, as we explained in Section 2.3.

**Varying the number of links** To guarantee the connectedness of the network (Theorem 2.1.1) we use at least 3 links per node. Figure 4.2 shows that increasing the number of links to 4 improves the efficiency slightly, but more important, it decreases the variance significantly. Increasing the number of links to higher values does not give any significant improvements.

Here, the mean in Figure 4.2 shows a clear resemblance with a horizontally flipped version (as the efficiency is 100% minus the error) of Figure 2.3.

**Varying the number of seed links** In Figure 4.3 we study the effect on the efficiency, as we vary the bandwidth of the seed, given as the number of links $L_{seed}$. Increasing this number up to the number of links all other
Figure 4.2: Bandwidth efficiencies with 1001 nodes with varying number of links.

Figure 4.3: Bandwidth efficiencies with varying number of seed links and 1000 nodes with 8 links.
nodes have, $L_{\text{node}}$, clearly improves the efficiency. After this point, only a very small increase is found. This can be explained with use of Corollary 2.1.2: the maximum flow is limited by the minimum of $L_{\text{seed}}$ and $L_{\text{node}}$.

**Heterogeneous number of links** In a real-world P2P network, nodes will have different bandwidths, leading, in our model, to a heterogeneous number of links per node. To study the effect of this, we have run simulations with the number of links uniformly spread between $12 - \Delta$ and $12 + \Delta$. Figure 4.4 shows that this has no effect on the efficiency.

### 4.2 Reliability

**Dynamic network** To study the effect of dynamic links on the efficiency, we have simulated a network with 1000 nodes, dropping and reattaching a number of links per timestep. When $L_{\text{seed}}$ equals $L_{\text{node}}$, there is a negative impact as seen in Figure 4.5. We can explain this with the intuitive approach we used in Section 3.2: nodes must receive all vectors over the $L$ shortest paths to the seed without delay. As a link is broken and reattached, a new shortest path is created. When this path is longer than the shortest path that came with the broken link, the vectors traveling over this path will arrive to the node with a delay and cause an inefficiency.

We can use the same solution for this as in Section 3.2. When we increase $L_{\text{seed}}$, there are more vectors in the network than the node has received. It can download these vectors while it is waiting for the delayed vectors. Figure 4.5 shows that this has no effect on the efficiency.

---

Figure 4.4: Bandwidth efficiencies with 1000 nodes with links uniformly spread between $12 - \Delta$ and $12 + \Delta$ and a seed with 20 links.
4.5 shows that a small increase of $L_{seed}$ already reduces the inefficiencies significantly. However, there can still be seen a small negative influence of the number of broken links on the efficiency.

**Leaving nodes** In a P2P environment it is not guaranteed that all nodes complete the full file. Users may choose to leave the network before finishing their download. Figure 4.6 studies the effect of users leaving the network uniformly spread, before completing. No significant effect on the efficiency is found up to 50% of the nodes leaving. We have slightly oversized $L_{seed}$ to prevent the dropped links to cause shortest path delays, as we have seen in Figure 4.5.

**Joining nodes** So far all simulations have started with all nodes joining the network at the same time ($t = 0$). Our final and most realistic simulation allows more and more nodes to join the network over time. Finished nodes will leave the network immediately after completing the download.

Figure 4.7 shows the efficiencies for an 8-link seed with 60 nodes with also 8 links joining the network each timestep. As the network grows over time, the number of joining nodes becomes relatively smaller with respect to the network size. Hence relatively fewer links have to be broken to allow these nodes to join, and therefore the efficiency increases, as Figure 4.5 showed us that, especially when $L_{seed} = L_{node}$, breaking links decreases the efficiency. Around timestep 4, another effect starts dominating the efficiency. The earliest joined nodes will start leaving the network, and thus as more and
more nodes leave the network over time, more links will be broken, and the efficiency will decrease. After timestep 12, the number of nodes leaving the network will equal the number of nodes that join the network, thus the network keeps the same size, and the efficiency stabilizes.

The second curve in Figure 4.7 shows the efficiencies for a slightly oversized seed (16 links) with 60 nodes with uniformly spread links between 4 and 12 joining the network each timestep. The earliest joined nodes now suffer less from the joining of new nodes, due to the oversized node. Again after a few timesteps, nodes start leaving the network, and the efficiency decreases. After timestep 14, the number of nodes leaving the network will equal the number of nodes that join the network, and the efficiency stabilizes.

Contrary to what we would expect based on Figure 4.4, we might conclude from Figure 4.7 that the stable efficiency of nodes with a heterogeneous number of links is lower than that of nodes with a homogeneous number of links, not equal. This difference is caused by two circumstances in this simulation that are different from the simulation of Figure 4.4. First, during the simulation nodes join and leave the network – leading to a number of links being broken and attached, depending on the number of links the nodes have. Second, the number of nodes in the network at the stable phase (when the number of nodes leaving equals the number joining) depends on the time it takes for them to finish, which in its turn depends on the number of links they have. As we have seen in Figures 4.1 and 4.5, both the size of the network and the number of links dropped per timestep influence the efficiency.

This means that there is no simple relation between a homogeneous or
homogeneous number of links, and the efficiency. A better insight of the influence of the size of the network and the number of links dropped on the efficiency, in a dynamic network, might provide us also with a better insight of the influence of a heterogeneous number of links.

### 4.3 Optimal Sparsity $\sigma$

In order to find the lowest sparsity $\sigma$ and seed bandwidth $L_{seed}$, where sparse network coding still performs as well as complete network coding, we use our model simulating 1001 nodes with 8 links each, varying both $\sigma$ and $L_{seed}$. As we use 100 vectors, sparse coding with $\sigma = 100$ is the same as complete network coding and can be used as the reference efficiency. Figure 4.8 shows that when $L_{seed} = L_{node}$, setting $\sigma$ to less than 100 immediately leads to a diminishing efficiency.

From the figure, we find that there is a trade-off between $L_{seed}$ and $\sigma$. In our opinion $L_{seed} = 10$ and $\sigma = 10$ offers the best general trade-off: the seed is oversized by 25%, while the encoding complexity is reduced by 90%. Of course in any network this decision should be made based on the available seed bandwidth and the computing power of the nodes. In [18] the optimal circumstances are found to be at $d = 10$, which, according to Equations 3.3 and 3.4, is similar to $\sigma = 12$. From now on when we speak of sparse network coding we implicitly mean $\sigma = 10$, unless stated otherwise.
Figure 4.8: Bandwidth efficiencies for sparse coding with varying sparsities with 1001 nodes with 8 links.

Figure 4.9: Bandwidth efficiencies for smart coding with varying randomness with 1001 nodes with [4,5,...,12] uniformly spread links.

4.4 Optimal Randomness $\rho$

In order to find the lowest randomness $\rho$ where smart network coding still performs as well as complete network coding, we use our model simulating 1001 nodes with [4,5,...,12] uniformly spread links each, varying $\rho$ between 1 and 100. As we use 100 vectors, smart coding with $\rho = 100$ is the same as complete network coding and can be used as the reference efficiency. Figure 4.9 suggests that setting $\rho$ to 2 is already enough for the new vectors to be “random enough”, as this leads to the same efficiency as setting $\rho$ to 100. In fact, in the figure, only $\rho = 1$ leads to a decreased efficiency.

As we verify that $\rho = 2$ is enough to perform as well as complete network coding with the more real-life simulations of Section 4.2; Figure 4.7 – with nodes joining and leaving each timestep – we see that indeed $\rho = 1$ is inferior
Table 4.1: Upper bounds for the coding complexities of various coding algorithms for 100 vectors and 8 links.

<table>
<thead>
<tr>
<th>Coding algorithm</th>
<th>Coding complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete coding</td>
<td>$\approx 150S$</td>
</tr>
<tr>
<td>Sparse coding</td>
<td>$\approx 108.55S$</td>
</tr>
<tr>
<td>Delta coding</td>
<td>$\approx 105.79S$</td>
</tr>
<tr>
<td>Smart coding</td>
<td>$\leq (100 + \rho)S$</td>
</tr>
</tbody>
</table>

to complete network coding (Figure 4.10). However, contradictory to Figure 4.9, $\rho = 2$ also performs less than complete network coding. It appears that in both Figures 4.10 and 4.11, as $\rho$ increases, the efficiency tends more and more towards complete network coding, which we predicted in Section 3.4. Due to the fluctuations in the efficiencies, this is not easy to see. When we average the values of all timesteps of Figure 4.10, we find

<table>
<thead>
<tr>
<th>Coding algorithm</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete network coding</td>
<td>98.8%</td>
</tr>
<tr>
<td>Smart-3 network coding</td>
<td>98.7%</td>
</tr>
<tr>
<td>Smart-2 network coding</td>
<td>98.6%</td>
</tr>
<tr>
<td>Smart-1 network coding</td>
<td>98.2%</td>
</tr>
</tbody>
</table>

Averaging the values of Figure 4.11, gives us

<table>
<thead>
<tr>
<th>Coding algorithm</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete network coding</td>
<td>98.72%</td>
</tr>
<tr>
<td>Smart-5 network coding</td>
<td>98.63%</td>
</tr>
<tr>
<td>Smart-4 network coding</td>
<td>98.59%</td>
</tr>
<tr>
<td>Smart-3 network coding</td>
<td>98.48%</td>
</tr>
</tbody>
</table>

which both show an increasing efficiency with a higher $\rho$.

We can see that under different circumstances, the deviation from the optimum is not the same for the same $\rho$. It also appears that the lower the efficiency of complete network coding is, the higher the $\rho$ of smart network coding has to be, to be “close” to this efficiency. Unfortunately, we do not have enough data to substantiate this conclusion.

### 4.5 Coding Algorithms Comparison

Using Equations 3.5, 3.6, 3.7, 3.8, and 3.9 we find the coding complexities $C$ of the coding algorithms for $V = 100$, $L = 8$, and $\sigma = 10$, summarized in Table 4.1.
Figure 4.10: Bandwidth efficiencies for smart coding with varying randomness with 60 nodes with 8 links that join in the corresponding timestep. The seed has 8 links.

Figure 4.11: Bandwidth efficiencies for smart coding with varying randomness with 60 nodes with [4,5,...,12] uniformly spread links that join in the corresponding timestep. The seed has 16 links.
To gain more insight in the performance of the other proposed coding algorithms, we re-use the simulation of Section 4.2; Figure 4.7 again, as we also did in Section 4.4.

Both Figures 4.12 and 4.13 show that delta network coding performs the same as complete network coding. Figure 4.12 shows that sparse network coding has some troubles with the first nodes that arrive to the network, due to the fact that $L_{\text{seed}} = L_{\text{node}}$, as we already predicted in Section 3.2. Indeed in Figure 4.13, where $L_{\text{seed}} > L_{\text{node}}$, the first nodes perform similar to complete network coding. We can see that as the efficiency stabilizes in a network with a heterogeneous number of links, sparse network coding performs slightly worse than complete network coding. It appears that sparse network coding is less well able to deal with networks with a heterogeneous number of links. Unfortunately, we do not have enough data to draw this conclusion.

Because delta network coding performs as efficient as complete network coding under all simulated circumstances, with a lower complexity than both complete and sparse network coding, it is superior to these algorithms. It has the disadvantage that it requires a delay between the reception and decoding of vectors.

As we have seen in Section 4.4, for small $\rho$, smart network coding performs worse than complete network coding (and thus also worse than delta network coding). On the other hand, these small $\rho$ offer a lower complexity than the complexity of delta network coding. In addition, it is only an upper bound, so the actual complexity could be even lower. Hence, smart network coding offers us a nice trade-off possibility: we can increase the efficiency by
increasing $\rho$, at the cost of a higher complexity and vice versa.

4.6 Comparison with Bittorrent and Naive Broadcast

We want to compare the obtained bandwidth efficiencies with the most used P2P file distribution program: Bittorrent. Unfortunately not much data is available on the efficiency of this program. Tian et al. [20] simulate a Bittorent-like network with a discrete-event simulator. The trace of a single node shows that the first and last 10% of the file cannot be downloaded using all available bandwidth, leading to a bandwidth efficiency of around 81%. Considering this, the network coding system, reaching 98 – 99%, seems very promising. However, we have to keep in mind that the simulations were not performed under the same circumstances.

Because M. Wang et al. [21] approximated Bittorrent to be 300% more efficient than naive broadcast, based on their observations, and we have found that network coding has the potential to outperform Bittorrent, it seems superfluous to make a comparison between the two. We can conclude from their work that nodes that use naive broadcast receive on average 386 fragments, before they are able to retrieve data that was split into 100 fragments. This leads to a bandwidth efficiency of 26% and subscribes to our estimated efficiency of Bittorrent, as 300% of 26% is approximately equal to the 81% we calculated from [20].

We believe that M. Wang et al. have found network coding to perform
very inefficient, due to the path they have followed in their search for the optimal parameters, as well as the large influence of the coding complexity on the average download time in their simulations. They start off with nodes that do not start forwarding vectors until they have collected at least 100 vectors. Clearly this leads to unnecessary high download times. As they vary the parameter “density” – which is essentially the same as our parameter $\sigma$ – they find that the download times decrease as the density decreases. They attribute this to the lower encoding complexity that comes with a smaller density, which has apparently an enormous influence on the download times. Perhaps this influence is caused because new vectors are not encoded on-the-fly in their system; this is not clear from their paper. They find that setting the density to 6%, similar to $\sigma = 6$, leads to the shortest download time, and use this setting in the following simulations. In our model, the coding complexity does not influence the efficiency, and in Figure 4.8 we can see that $\sigma = 6$ does not lead to a very high bandwidth efficiency.

As they realize that having the nodes wait until they have received 100 vectors is not optimal, they now vary the number of vectors a node must have received before it starts sending. They call this the “aggressiveness”. Indeed lower values for the aggressiveness lead to shorter download times. However, as the density is now set to 6%, they never find network coding to outperform naive broadcast. We believe that, would they have set the density to 100%, set the aggressiveness to 1 vector, and implemented an on-the-fly encoding algorithm, they would have found network coding to be better performing than in their existing simulations.
Chapter 5

Implementation Issues

Our model of a P2P system makes use of a lot of simplifications. In this chapter we discuss some of the aspects of a network coding environment that have not (accurately) been simulated.

5.1 Complexity

As we have calculated in Chapter 3, the coding complexity of network coding is of the order $V \cdot S$. Not only does the use of more vectors lead to a higher complexity, it also causes more overhead as the encoding vector gets larger. Thus, from the computing complexity and overhead point of view, we wish to keep the number of vectors as low as possible.

Larger vectors lead to higher forwarding delays as nodes wait until they have received the total vector before they will use it in encoding new ones. We could of course force the encoding process of the nodes to use vectors they have not yet fully received. However, in that case, when a node is not able to finish the downloading of a vector (most probably due to a failure in the link with the sending neighbor), it will have to cancel the sending of all encoded vectors, as it can never finish the encoding of these vectors. The effect of this disadvantage is amplified by another consequence of increasing the vector size: the probability for a link failure during the download of a vector increases as well. Thus, from the network layer point of view, we wish to keep the vectors small.

Group network coding  So, how do we manage to obtain a low number of small vectors, when the size of our data is large (e.g. in the distribution of large files)? In [13, 11, 12] Gkantsidis et al. have solved this using group network coding, based on the generations system of [6]. The data is first split into several groups, which in their turn are split into vectors. Only vectors from the same group are combined into a new vector. The node now still has to use fragment scheduling to decide from which group it is going
to download a vector. For this it could use the local-rarest-first scheme, or even simply download a vector from the group it has downloaded the least vectors from.

For streaming media we could also use group network coding, but now nodes can simply download the groups chronologically. As soon as a group is totally downloaded, it can be decoded and played. Annapureddy et al. even propose a Video on Demand service [2] that combines group network coding with group scheduling and topology management.

It is easy to see that group network coding also comes with a trade-off: consider the situation where each group only contains one vector. This would bring the computing complexity to a minimum, but results in a store-and-forward system. It would be an interesting research topic to find the optimal ratio between the number of groups and the number of vectors.

5.2 Spread Delay

As we have seen in Section 4.1, for a connected network, we can choose the number of links $L$ freely, as long as nodes have at least 3 links. However, a minimum of 4 links would be preferable, as this decreases the variance of the bandwidth efficiencies significantly. The spread delay of the vectors through the network depends on the number of links, and we wish to keep this delay as low as possible. Having more links leads to a smaller average hopcount (see Appendix A), but also causes each link to have a smaller bandwidth $B_{W_{link}}$, as the total bandwidth $B_{W_{tot}}$ will remain constant.

\[ B_{W_{link}} = \frac{B_{W_{tot}}}{L} \] (5.1)

Combining Equations 5.1 and A.1 we find an expression for the spread delay

\[ E[t_{delay}] = \frac{E[H_{N,L}]}{B_{W_{link}}} = \frac{1.08 \cdot \log N}{B_{W_{tot}}} \cdot \frac{L}{\log L} \]

which has its minimum at $L = e = 2.718$. Instructing nodes to keep their number of links as close to 4 as possible, while still reaching their required bandwidth, should thus give the lowest spread delay.

It would be interesting to perform real-live tests to see whether 4 links are (generally) enough to reach the required (or maximum) bandwidth. This would be more likely to happen when all nodes require the same bandwidth (e.g. with streaming media) than when nodes simply want to download at maximum speed (e.g. with file downloads), which differs depending on their connection.
5.3 Incentive Mechanisms

As we have seen in Section 1.1, for the incentive mechanism not to stall our download, it is important that both ends of a link have something interesting to offer. From Figure 4.7 we find the probability that the other end has something interesting between 98% and 99%. The probability that we cannot exchange vectors with a random neighbor is then $1-(0.98)^2 \approx 0.04$. This means that with 4 or 8 links, the probability that we cannot exchange with any neighbor, is respectively $(0.04)^4 = 2.6 \cdot 10^{-6}$ or $(0.04)^8 = 6.55 \cdot 10^{-12}$.

Jun and Amahad [16] propose an incentive mechanism that is fairer than Bittorrent. However, the overall efficiency of their mechanism is worse than Bittorrent’s mechanism. They blame this on the inefficient fragment scheduling scheme that is used in Bittorrent (local-rarest-first). Since we have shown that network coding could offer more efficient fragment scheduling than Bittorrent (i.e., a higher bandwidth efficiency), this could also improve the efficiency of the proposed incentive mechanism.

Where free-riders in a store-and-forward system try to minimize the amount of data they upload, while still being able to download, with network coding, a new type of free-riders might surface. As network coding involves a lot of computations, free-riders can try to minimize the number of computations their computer has to run. In the decoding process there is not much (if anything) to gain, as they cannot influence the form in which the data arrives and this data has to be decoded to retrieve the original data. However, the encoding process is vulnerable to free-riders: they could simply “make up” new vectors, or not combine any vectors but send a previously received one. We could prevent the former method with the security system discussed in Section 5.4, but the latter, which reduces network coding to the traditional store-and-forward system, cannot be prevented.

This provides us with an even stronger incentive to use a coding algorithm that has a low complexity encoding process. When the complexity of the decoding process is much larger than that of the encoding process, the gain free-riders have will not disappear, but at least be much smaller, which makes it less interesting to free-ride.

5.4 Security

In any public P2P system we need to account for the possibility that malicious users can produce “wrong” data. In a classic store-and-forward system, we can take care of this by digitally signing every block of data. However, since in a network coding system each node produces new data (combining the received vectors), we cannot use this method. Gkantsidis and Rodriguez [14] propose a system of secure random checksums (SRC’s), which they claim
can be computed very efficiently and offers a high security. This would solve an important issue of network coding; although we have not looked at security in this report, it is an essential part of any proposed network coding protocol. The warranty that the received data is valid will make or break its (commercial) success.
Chapter 6

Conclusions

6.1 Conclusions

As P2P content distribution systems have become very popular over the last years, it has drawn a lot of attention of researchers. A first look at the use of network coding in such systems seems promising: it could improve download speeds and facilitate incentive mechanisms very well. Existing literature is contradictory – concluding on one hand that network coding can outperform Bittorrent 2-3 times and on the other hand that it is inferior to naive broadcast (a worst-case protocol).

To be able to compare the performance of network coding with Bittorrent and naive broadcast under several circumstances, we investigated the bandwidth efficiency of network coding in a content distribution system. We used a simulation model for a P2P network, consisting of (many) nodes and one registrar. Nodes could exchange information with a small subset of the nodes in the network (their neighbors). They were all connected to the registrar, to which they reported their statistics and requested new neighbors.

From our simulations we concluded that the bandwidth efficiency of network coding in a P2P content distribution system varies between 98 and 99%. Small fluctuations in the efficiency are caused by

- The network size
- The number of links to neighbors per node (it must have at least 3, preferably 4)
- The seed’s bandwidth (as long as it is at least equal to the nodes’ bandwidth)
- Dropped links (e.g., caused by nodes joining and leaving the network)

The most important cost of network coding is that it requires a lot of computing power. To reduce the complexity, we have proposed two new coding algorithms:
• Delta network coding – Nodes only use the last received vectors when they create new vectors for their neighbors. The algorithm performs better in terms of bandwidth efficiency and complexity to the existing algorithms complete network coding and sparse network coding. It does come with a delay between the reception and decoding of vectors, which could make it less interesting to time-critical applications, such as streaming video.

• Smart network coding – Nodes only use the last $\rho$ decoded vectors when they create new vectors for their neighbors. The algorithm offers a trade-off between complexity and bandwidth efficiency with its parameter $\rho$. We have only found an upper bound for this algorithm’s complexity.

Based on results of [20] and [21] we have calculated that Bittorrent reaches a bandwidth efficiency of 81% and naive broadcast of 26%. From this we could conclude that network coding outperforms both protocols. However, we have to keep in mind that those simulations were performed under different circumstances than ours.

Finally, we emphasize that not all aspects of a real-world implementation of a network coding environment have been covered by our simulations. Some of these would make interesting subjects of further research.

6.2 Directions for Further Research

We have looked into the influence of some basic properties such as the network size and number of links per node, in a static environment (with no nodes leaving or joining the network), on the efficiency of network coding. The simulations in a dynamic environment showed that these basic properties can have different influences under dynamic circumstances (compare Figure 4.4 with Figure 4.7). More research on network coding in dynamic networks might provide new insights.

Discrete-time simulations that also consider the information vectors could give us more insight in the coding complexity of smart network coding.

A real-world implementation could provide us with more evidence that network coding can indeed reach a bandwidth efficiency of 98–99%, and we can also see whether the relatively small number of required links (3 or 4) is really enough to reach this efficiency.

Extending our model for the use of group network coding would give us the possibility to find the optimal ratio between the number of groups and the number of vectors within those groups as well as a well-performing group scheduling scheme.
Appendix A

The Hopcount in Random Regular Graphs

Throughout this report we have assumed that the expected value of the hopcount between an arbitrary pair of nodes increases when the size of the network increases or the number of links decreases, and vice versa. To prove this, and also to be able to find the number of links that minimize the spread delay (see Section 5.2), we want to find the relationship between the number of nodes and links, and the average hopcount. Kim [17] uses some results of [19] to asymptotically approximate the average of the hopcount $H_{N,L}$ in a large random regular graph $\mathcal{G}(N, L)$ as

$$E[H_{N,L}] = \frac{\log[(N - 1)(L^2 - L) + L^2] - \log L^2}{\log L} \propto \log_{L} N$$

To verify this model of the average hopcount we have calculated the average hopcount from simulations where we vary both $N$ and $L$. Figures A.1 and A.2 show that

$$E[H_{N,L}] = 1.08 \cdot \log_{L} N \quad (A.1)$$

approximates the average hopcount of random regular graphs well for any $N$ and $5 \leq L \leq 15$. 
Figure A.1: Simulations and model with varying number of links and 1001 nodes.

Figure A.2: Simulations and model with varying number of nodes and 8 links.
Nomenclature

$A_L$ The local encoding matrix
$A$ The global encoding matrix
$E$ or $E_i$ A global encoding vector
$I$ or $I_i$ An information vector
$M_i$ One of the $V$ fragments $M$ is split into
$M$ The total original data
$R$ A local encoding vector
$\mathcal{F}$ A finite field (or Galois field)
$\mathcal{G}(N, L)$ A random regular graph with $N$ nodes and $L$ links per node
$\rho$ Randomness: the number of last received vectors used in the encoding process [Smart network coding]
$\sigma$ Sparsity: the number of received vectors used in the encoding process [Sparse network coding]
$B_{W_{eff}}$ The effectively available bandwidth
$B_{W_{link}}$ Bandwidth of a link between two nodes
$B_{W_{used}}$ The used bandwidth
$BE$ Bandwidth efficiency
$C$ The total coding complexity
$C_d$ The decoding complexity
$C_e$ The encoding complexity
$H_{N,L}$ The hopcount between two random nodes in $\mathcal{G}(N, L)$
$L$ The number of links per node
\[ l \quad \text{The length of the information vectors} \]
\[ N \quad \text{The number of nodes} \]
\[ p \quad \text{The probability that a received vector is used in the process of encoding a new vector} \text{ [Sparse network coding]} \]
\[ r \quad \text{A (random) coefficient} \]
\[ S \quad \text{The total size of the data} \]
\[ V \quad \text{The number of vectors} \ \mathbf{M} \text{ is split into} \]
\[ V_{inn} \quad \text{The number of received innovative vectors} \]
\[ V_{tot} \quad \text{The number of received vectors} \]
\[ X_i \quad \text{The number of cycles of length} \ i \ \text{in} \ \mathcal{G}(N, L) \]
Bibliography


